Clase 4 — 21.11.25

#VSC #javascript

👩‍🏫 Profesora: Sara Gonzalo
📘 Desarrollo de interfaces. JAVASCRIPT, JQUERY, REALIDAD VIRTUAL
🗓️ Clase 4 — 21/11/2025
🎯 **Tema:** Final Continuacion Java Script | DOM

1️⃣2️⃣ Bucles en JavaScript

Los bucles permiten repetir instrucciones múltiples veces. Sara recalca que en JavaScript funcionan igual que en Java o C, a diferencia de Python.

🔹 Bucle for (cuando sabemos cuántas veces repetir)

```
for (let contador = 0; contador < 10; contador++) {
    // instrucciones
}
```

Características:

- Usa **contador**, **condición** y **actualización**.
- Es el bucle más utilizado cuando el número de iteraciones está claro.
- Estructura idéntica a lenguajes clásicos (Java/C/C++).

Ejemplo visto en clase:

```
for (let contador = 1; contador < 11; contador++) {
    document.write("El valor de contador es: " + contador + "<br>");
}
```

Sara recuerda que `document.write` es solo para practicar, **no se usa en proyectos reales**.

1️⃣3️⃣ Bucle while

El while es **más restrictivo**:

- Solo entra si la condición ya es verdadera.
- Si no se cumple, el bloque **no se ejecuta ni una sola vez**.

```
let num = 0;

while (num != 0) {
    window.alert("Dentro del bucle!!!");
}
```

1️⃣4️⃣ Bucle do…while

Este bucle se creó para resolver la limitación del while:

✔ entra al menos **una vez**
✔ luego evalúa la condición

Se usa cuando **es obligatorio ejecutar algo antes de comprobar**.

```
let num2 = 0;

do {
    window.alert("Dentro del bucle do while");
} while (num2 != 0);
```

1️⃣5️⃣ Arrays en JavaScript

Un **array** es una estructura de datos que permite almacenar **múltiples valores** en una sola variable, organizados en celdas indexadas numéricamente.

Características clave:

- Los índices comienzan en **0** → la primera posición es `0`, la última es `length - 1`.
- Son **dinámicos** → pueden crecer o reducirse durante la ejecución del programa.
- Pueden almacenar **valores de distintos tipos** (algo que no ocurre en lenguajes fuertemente tipados como Java o C#).
- Internamente, un array es un **objeto especial** con propiedades y métodos propios.

Ejemplo:

```
let alumnos = ["pedro", "maria", "eva", "lucia", 23, true, false];
```

El array anterior mezcla:

- strings
- números
- booleanos

Esto es válido porque JavaScript no exige un tipo homogéneo.

Recomendación de Sara (muy importante):

Declarar arrays sin número de posiciones:

```
let tabla = [];
```

Motivos:

- Evita errores si el array crece más de lo previsto.
- Es más flexible cuando los datos provienen de formularios, cálculos o API externas.
- Permite trabajar con arrays “en crudo”, tal y como se usan en la industria.

1️⃣6️⃣ Acceso, modificación y longitud

🔹 Acceder a un valor

Usamos el índice entre corchetes:

```
alumnos[0]   // "pedro"
alumnos[3]   // "lucia"
```

🔹 Modificar un valor

```
alumnos[0] = "Luis";
```

Cuando modificas, **no cambias el tamaño del array**, solo el contenido de esa celda.

🔹 Longitud del array

```
alumnos.length
```

Devuelve la cantidad de celdas existentes.

Sara remarcó el error más común de los estudiantes:

❌ `lenght`
✔️ `length`

Detalle importante:

La propiedad `length` **no es inmutable**.
Si asignas un índice superior, el array crece automáticamente:

```
let datos = [];
datos[5] = "hola";

console.log(datos.length);  // 6
```

Las posiciones intermedias quedan como `undefined`.

1️⃣7️⃣ Métodos de arrays

Los arrays tienen métodos muy útiles para manipular datos:

`push()`

Añade un elemento **al final**.

```
frutas.push("kiwi");
```

`pop()`

Elimina y devuelve el **último elemento**.

```
let ultimo = frutas.pop();
```

`shift()`

Elimina y devuelve el **primer elemento**.

```
frutas.shift();
```

`unshift()`

Añade un elemento **al principio**.

```
frutas.unshift("fresa");
```

Otros métodos los verás cuando lleguemos al DOM y JSON (`splice` , `slice` , `map` , `filter` , etc.), pero Sara se centra por ahora en los más básicos.

1️⃣8️⃣ Recorrer arrays

Recorrer un array significa **visitar cada celda una por una** para leer su valor, modificarlo o usarlo en algún cálculo.

🔹 For clásico (Sara: “el más eficiente y controlado”)

```
for (let i = 0; i < frutas.length; i++) {
    console.log(frutas[i]);
}
```

Ventajas:

- Máximo rendimiento.
- Control completo del índice.
- Recorre exactamente las posiciones válidas.

Es el método estándar para arrays grandes.

🔹 For…in

```
for (let celda in frutas) {
    console.log(celda);  // imprime el índice
}
```

Sara advierte:

- Es **más lento**, porque recorre *propiedades enumerables* del objeto, no solo índices.
- Puede recorrer claves añadidas manualmente.
- No es adecuado para arrays grandes.

Se utiliza más en objetos, no en arrays.

1️⃣9️⃣ Arrays multidimensionales (Matrices)

Una **matriz** es un array cuyos elementos son otros arrays. En JavaScript se usan para representar **datos tabulares** y para **volcar información de fuentes externas** (APIs, JSON, bases de datos).

📌 Cuándo usar matrices

- Tablas (notas, inventarios, horarios).
- Datos estructurados por filas/columnas.
- Respuestas JSON con listas de listas.
- Preparación de datos para renderizar en DOM (tablas HTML).

🔹 Ejemplo 1: Tabla de notas

```
let notas = [
    ["Ana", 7, 8, 9],
    ["Luis", 6, 5, 7],
    ["Eva", 9, 8, 10]
];
```

- `notas[i]` → fila (un alumno).
- `notas[i][0]` → nombre.
- `notas[i][1..n]` → calificaciones.

Recorrido:

```
for (let i = 0; i < notas.length; i++) {
    for (let j = 1; j < notas[i].length; j++) {
        console.log(notas[i][0], "nota:", notas[i][j]);
    }
}
```

🔹 Ejemplo 2: Inventario simple

```
let inventario = [
    ["Teclado", 15, true],
    ["Ratón", 30, false],
    ["Pantalla", 8, true]
];
```

Estructura típica:

- `[nombre, stock, disponible]`

🔹 Ejemplo 3: Matriz desde datos externos (JSON simulado)

```
let datosAPI = [
    [101, "Pedido A", "Enviado"],
    [102, "Pedido B", "Pendiente"],
    [103, "Pedido C", "Cancelado"]
];
```

Este formato es habitual al **consumir APIs** y luego **pintar tablas** con DOM.

🧠 Patrón mental para matrices (muy importante)

- **Primer índice** → estructura principal (fila).
- **Segundo índice** → detalle (columna).
- **Dos bucles** siempre:
  - externo = filas
  - interno = columnas

```
for (let i = 0; i < matriz.length; i++) {
    for (let j = 0; j < matriz[i].length; j++) {
        // matriz[i][j]
    }
}
```

Este patrón se reutiliza **tal cual** cuando trabajes con DOM.

📝 Ejercicios tipo examen

✅ Ejercicio 1

Dada la matriz:

```
let m = [
    [1, 2],
    [3, 4],
    [5, 6]
];
```

Pregunta:
¿Cuántas filas tiene? ¿Cuántos elementos en total?

Respuesta esperada:

- Filas: `m.length` → 3
- Elementos: suma de `m[i].length` → 6

✅ Ejercicio 2

Recorre la matriz y muestra solo los números pares.

```
for (let i = 0; i < m.length; i++) {
    for (let j = 0; j < m[i].length; j++) {
        if (m[i][j] % 2 === 0) {
            console.log(m[i][j]);
        }
    }
}
```

✅ Ejercicio 3 (muy típico)

Suma todos los valores de una matriz.

```
let suma = 0;

for (let i = 0; i < m.length; i++) {
    for (let j = 0; j < m[i].length; j++) {
        suma += m[i][j];
    }
}

console.log(suma);
```

🔗 Conexión directa con el DOM (adelanto)

Sara enlaza este tema con lo siguiente:

- El **DOM es una estructura jerárquica**, igual que una matriz.
- Aprender a recorrer matrices te prepara para:
  - recorrer nodos
  - acceder a hijos
  - crear elementos dinámicos

👉 **Matrices → DOM → Páginas generadas solo con JavaScript**

2️⃣0️⃣ Comentarios finales de Sara

- **while → más restrictivo**, depende completamente de la condición inicial.
- **do…while → garantiza ejecutar al menos una vez**.
- **arrays → declararlos vacíos y llenarlos según el flujo del programa**.
- **matrices → fundamentales cuando consumamos datos externos**.
- Aquí termina la parte de lógica; **en la siguiente clase comenzamos DOM**.

2️⃣1️⃣ Ejercicios de operadores lógicos (tarea obligatoria)

Estos dos ejercicios consolidan el uso de:

- `prompt()` para recoger datos
- Operadores lógicos (`!` , `&&`)
- Condicionales `if / else if / else`
- Validaciones básicas de entrada
- Control de flujo según condiciones encadenadas

Se trabajaron como **tarea obligatoria**, pero encajan perfectamente dentro del contenido de la Clase 4.

🟦 Ejercicio 1 — Uso del operador NOT (`!`)

🎯 Objetivo

- Pedir un número por pantalla.
- Validar que sea **mayor que cero** y **distinto de cero**.
- Si es válido, indicar si es **par** o **impar**.
- Si no es válido, mostrar un mensaje de error.

📌 Código JavaScript comentado

```
// Pedimos un número entero al usuario mediante prompt.
// parseInt convierte el texto introducido en un número entero.
let numero = parseInt(window.prompt("Introduce un número"));

// Validamos que el número sea mayor que cero.
// Usamos el operador lógico NOT (!) para negar la condición "numero <= 0".
// Si "numero <= 0" es true, el NOT lo convierte en false.
// Si "numero <= 0" es false (es decir, número positivo), entonces entra en el if.
if (!(numero <= 0)) {

    // Determinamos si el número es par usando el operador módulo (%).
    // Un número es par si el resto al dividir entre 2 es 0.
    if (numero % 2 === 0) {
        window.alert("El número es PAR");
    }
    // Si no es par, será impar.
    else {
        window.alert("El número es IMPAR");
    }

}
// Si no se cumple la validación principal (mayor y distinto de cero), mostramos
error.
else {
    window.alert("Error: el número debe ser mayor que cero y distinto de cero");
}
```

🧠 Puntos clave reforzados por Sara

- El operador **NOT** es útil cuando queremos negar **condiciones completas**, no valores sueltos.
- Antes de hacer cálculos, siempre validar los datos introducidos por el usuario.
- `prompt()` devuelve texto → siempre convertir a número (`parseInt` , `parseFloat`).

🟦 Ejercicio 2 — Condicionales anidadas y validación por rangos

🎯 Objetivo

1. Pedir un número entero.
2. Comprobar que sea **positivo y distinto de cero**.
3. Si es:
    - **De dos cifras** → indicar si es par o impar.
    - **De tres cifras** → mostrar el resto de dividir entre 2.

4. Si no cumple ninguna condición → error.

📌 Código JavaScript comentado

```
// Solicitamos un número entero al usuario.
let num = parseInt(window.prompt("Introduce un número entero"));

// Primera validación global: el número debe ser positivo y distinto de cero.
// Si no cumple, se muestra error automáticamente.
if (num > 0) {

    // --------- Caso 1: Número de dos cifras ---------
    // Comprobamos si está entre 10 y 99 (ambos incluidos).
    if (num >= 10 && num <= 99) {

        // Determinamos si es par o impar.
        if (num % 2 === 0) {
            window.alert("Número de dos cifras y PAR");
        } else {
            window.alert("Número de dos cifras e IMPAR");
        }
    }

    // --------- Caso 2: Número de tres cifras ---------
    // Entre 100 y 999.
    else if (num >= 100 && num <= 999) {

        // Calculamos el resto de dividir entre 2.
        window.alert("Resto al dividir entre 2: " + (num % 2));
    }

    // --------- Caso 3: No es de dos ni tres cifras ---------
    else {
        window.alert("Error: el número no es de dos ni de tres cifras");
    }

// Si el número no es positivo o es cero, mensaje de error directo.
} else {
    window.alert("Error: el número debe ser positivo y distinto de cero");
}
```

🧠 Conceptos importantes del ejercicio

🔹 Rangos numéricos

Para identificar si un número es de dos o tres cifras:

- Dos cifras → `10 ≤ num ≤ 99`
- Tres cifras → `100 ≤ num ≤ 999`

🔹 Operador AND (`&&`)

Se usa cuando dos condiciones deben cumplirse simultáneamente:

```
num >= 10 && num <= 99
```

🔹 Operador módulo (`%`)

Muy utilizado en programación inicial:

- Para comprobar paridad
- Para obtener restos en divisiones
- Para patrones numéricos

📝 Conclusiones didácticas según Sara

- Estos ejercicios entrenan la **estructura mental** básica para trabajar posteriormente con el **DOM**, donde validar datos será imprescindible.
- Los operadores lógicos se usan para **dirigir el flujo de ejecución** según reglas muy claras.
- Es importante escribir condiciones **legibles**, bien organizadas y sin redundancias.

2️⃣2️⃣ Ejercicios de Arrays

> En este apartado se aplican los conceptos básicos de arrays vistos en clase mediante ejercicios prácticos.
> Se trabaja la **creación**, **recorrido**, **acceso por índice** y la **conexión con el DOM**, utilizando estructuras y sintaxis propias de JavaScript básico.

🎯 Objetivo general

- Consolidar el uso de arrays en JavaScript.
- Practicar bucles `for` y `while`.
- Entender cómo mostrar resultados tanto en **consola** como en el **documento HTML**.
- Introducir la relación entre **arrays y DOM** de forma progresiva.

✅ Ejercicio 1 — Array dinámico y recorrido con bucles

🔹 Enunciado

- Se pide por pantalla el número de celdas del array.
- Cada celda guarda el número correspondiente a su posición.
- Imprimir el array:
  - con un bucle `for`
  - con un bucle `while`
- Mostrar el resultado en el documento.

🧩 Código JavaScript (archivo enlazado)

```
const salida = document.getElementById("salida");
const N = parseInt(prompt("Introduce el número de celdas del array:"), 10);
```

```
if (Number.isNaN(N) || N <= 0) {
  salida.textContent = "Valor no válido. Debe ser un número entero mayor que 0.";
} else {
  const array = new Array(N);

  // Se rellena el array con la posición de cada elemento
  for (let i = 0; i < array.length; i++) {
    array[i] = i;
  }

  // Recorrido con bucle for
  salida.textContent += "Recorrido con for:\n";
  for (let i = 0; i < array.length; i++) {
    salida.textContent += array[i] + "\n";
  }

  // Recorrido con bucle while
  salida.textContent += "\nRecorrido con while:\n";
  let i = 0;
  while (i < array.length) {
    salida.textContent += array[i] + "\n";
    i++;
  }
}
```

🔍 Análisis de funciones y métodos usados

`prompt()`

- Muestra una ventana emergente al usuario.
- Permite introducir datos por pantalla.
- Devuelve un `string` o `null`.

`parseInt(valor, 10)`

- Convierte el texto introducido en un número entero.
- El segundo parámetro (`10`) fuerza la base decimal.

`Number.isNaN()`

- Comprueba si el valor no es un número válido.
- Se usa para validar la entrada del usuario.

`new Array(N)`

- Crea un array con `N` posiciones.
- Las posiciones quedan vacías hasta ser rellenadas.

`array.length`

- Propiedad que indica el número de elementos del array.

- Se utiliza como límite en los bucles.

`textContent`

- Permite escribir texto dentro de un elemento HTML.
- Es seguro y no interpreta HTML.
- Combinado con `<pre>` respeta los saltos de línea.

Perfecto. A continuación tienes **el apartado 2️⃣2️⃣ – Ejercicio 2 reescrito y ampliado en profundidad**, con **explicaciones largas**, **línea a línea**, **métodos**, **razones de diseño** y **puntos clave remarcados**, tal y como suele valorar Sara en correcciones y exámenes.

Puedes **sustituir directamente** el bloque del *Ejercicio 2* que ya tienes por este.

✅ Ejercicio 2 — Array de imágenes y visualización dinámica en el documento

🔹 Enunciado

- Diseñar un array con 5 imágenes.
- Preguntar al usuario cuál desea imprimir.
- Una vez que se teclea el nombre de la imagen, esta **aparece en el documento HTML**.

Este ejercicio introduce por primera vez una **interacción completa entre JavaScript, arrays y el DOM**, y es clave para entender cómo los datos influyen en lo que se muestra en pantalla.

🎯 Objetivo didáctico del ejercicio

Este ejercicio persigue varios objetivos simultáneos:

- Trabajar con **arrays más complejos** (arrays de objetos).
- Practicar la **búsqueda de información dentro de un array**.
- Aprender a **validar datos introducidos por el usuario**.
- Introducir la **manipulación del DOM**:
  - cambiar atributos HTML
  - mostrar u ocultar elementos
- Entender el flujo:
  usuario → dato → lógica → resultado visual

🧱 Estructura de datos utilizada

Array de objetos

```
const imagenes = [
  { nombre: "sol", archivo: "img/sol.jpg" },
  { nombre: "luna", archivo: "img/luna.jpg" },
  { nombre: "tierra", archivo: "img/tierra.jpg" },
  { nombre: "marte", archivo: "img/marte.jpg" },
```

```
  { nombre: "jupiter", archivo: "img/jupiter.jpg" }
];
```

🔍 Explicación en profundidad

- `imagenes` es un **array**.
- Cada posición del array contiene un **objeto**.
- Cada objeto representa una imagen y tiene dos propiedades:
  - `nombre`: texto que el usuario debe escribir.
  - `archivo`: ruta real del fichero de imagen.

📌 Punto clave
Este patrón es mucho más potente que un array simple de strings, porque separa:

- el **dato lógico** (nombre)
- del **dato técnico** (ruta del archivo)

Este mismo esquema se usa en:

- inventarios
- listados de productos
- datos cargados desde JSON
- aplicaciones reales

🧩 Obtención de elementos del DOM

```
const mensaje = document.getElementById("mensaje");
const img = document.getElementById("imagen");
```

Qué hace cada línea

`document.getElementById()`

- Busca un elemento HTML por su atributo `id`.
- Devuelve una referencia directa al nodo HTML.
- Permite modificar el contenido o atributos desde JavaScript.

En este ejercicio:

- `mensaje` se usa para mostrar texto informativo al usuario.
- `img` se usa para mostrar u ocultar la imagen seleccionada.

📌 Punto clave
Sin esta referencia al DOM, JavaScript no podría modificar la página.

🧾 Entrada de datos del usuario

```
const opcion = prompt("¿Qué imagen deseas imprimir?");
```

Funcionamiento de `prompt()`

- Muestra una ventana emergente.
- Permite al usuario introducir texto.

- Devuelve:
  - un `string` si el usuario escribe algo
  - `null` si pulsa cancelar

📌 **Importante**
`prompt()` solo funciona en navegador, no en Node.js.

🔎 Normalización del texto introducido

```
const pedido = opcion.trim().toLowerCase();
```

Métodos usados sobre strings

`trim()`

- Elimina espacios en blanco al inicio y al final.
- Evita errores como:
  - `" luna"`
  - `"luna "`

`toLowerCase()`

- Convierte todo el texto a minúsculas.
- Permite comparar sin importar si el usuario escribe:
  - `LUNA`
  - `Luna`
  - `luna`

📌 **Punto clave**
La normalización del dato es esencial para evitar fallos de comparación.

🔁 Búsqueda dentro del array

```
for (let i = 0; i < imagenes.length; i++) {
  if (imagenes[i].nombre === pedido) {
    ...
    break;
  }
}
```

Análisis del bucle

- Se utiliza un **`for` clásico** porque:
  - se necesita el índice
  - es el método más controlado
- `imagenes.length` marca el límite del recorrido.
- `imagenes[i]` accede al objeto actual.
- `imagenes[i].nombre` accede a la propiedad `nombre`.

Comparación

```
imagenes[i].nombre === pedido
```

- Se usa `===` (comparación estricta).
- Compara valor y tipo.
- Es la forma recomendada en JavaScript moderno.

🛑 Uso de `break`

```
break;
```

- Detiene el bucle inmediatamente.
- Evita seguir recorriendo el array innecesariamente.
- Mejora la eficiencia y la claridad del código.

📌 **Punto clave**
Una vez encontrada la imagen, no tiene sentido seguir buscando.

🖼️ Mostrar la imagen en el documento

```
img.src = imagenes[i].archivo;
```

Qué ocurre aquí

- `src` es el atributo HTML del `<img>`.
- Al asignarlo:
  - el navegador carga la imagen indicada
  - si la ruta es correcta, la imagen aparece

📌 Si la imagen no existe o la ruta es incorrecta, el `<img>` queda vacío.

👁️ Control de visibilidad

```
img.style.display = "block";
```

y

```
img.style.display = "none";
```

Explicación

- `style.display` controla si un elemento se muestra o no.
- `"none"`:
  - oculta completamente el elemento
  - no ocupa espacio en la página
- `"block"`:
  - muestra el elemento
  - ocupa espacio

📌 **Punto clave**
Esto evita que aparezca una imagen vacía si el usuario se equivoca.

🗒️ Mensajes al usuario

```
mensaje.textContent = "Mostrando imagen: " + imagenes[i].nombre;
```

`textContent`

- Inserta texto plano en el HTML.
- No interpreta etiquetas HTML.
- Es más seguro que `innerHTML`.

Se usa para:

- informar al usuario
- mostrar errores
- mejorar la experiencia de uso

❌ Gestión del error (imagen no encontrada)

```
if (!encontrada) {
  mensaje.textContent =
    "La imagen indicada no existe o no se ha introducido ninguna.";
  img.style.display = "none";
}
```

Explicación

- `encontrada` actúa como **bandera de control**.
- Si nunca se cambia a `true`, significa que:
  - el usuario escribió un nombre incorrecto
  - o canceló el `prompt`

📌 **Punto clave**
Separar la lógica de búsqueda del control de errores hace el código más claro.

🧠 Ideas clave que hay que recordar (muy importantes)

- Un **array de objetos** permite representar datos complejos.
- Siempre hay que **normalizar la entrada del usuario**.
- El DOM se manipula mediante:
  - `getElementById`
  - propiedades (`src`, `textContent`)
  - estilos (`style.display`)
- `break` optimiza bucles de búsqueda.
- Este ejercicio es la base de:
  - formularios
  - listados dinámicos

- interfaces interactivas

📘 Tarea Switch — Desarrollo de Interfaces (Viajes con JavaScript)

🗂️ 1) Estructura del proyecto

```
proyecto/
 ├─ index.html
 ├─ app.js
 └─ img/
     ├─ nope.jpg
     ├─ paris1.jpg
     ├─ paris2.jpeg
     ├─ paris3.jpg
     ├─ marsella1.jpeg
     ├─ marsella2.jpeg
     ├─ marsella3.jpeg
     ├─ lyon1.jpeg
     ├─ lyon2.jpeg
     ├─ lyon3.jpeg
     ├─ playa1.jpeg
     ├─ playa2.jpeg
     ├─ playa3.jpg
     ├─ montana1.jpeg
     ├─ montana2.jpg
     ├─ lisboa.jpeg
     ├─ algarve1.jpg
     ├─ algarve2.jpg
     ├─ nazare.jpg
     └─ fatima.jpeg
```

🧾 2) `index.html` (código tal cual)

```
<!doctype html>
<html lang="es">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width,initial-scale=1" />
  <title>Tarea Switch — Desarrollo de Interfaces</title>
  <style>
    body{font-family:system-ui,Arial,sans-serif;margin:24px;line-height:1.5}
    header{max-width:900px}
    .panel{max-width:900px;border:1px solid #ddd;border-
radius:12px;padding:16px;margin-top:16px}
    .imgs{display:flex;flex-wrap:wrap;gap:12px;margin-top:12px}
    .imgs img{width:220px;max-width:100%;height:auto;border-radius:10px;border:1px
solid #eee}
    button{padding:10px 14px;border-radius:10px;border:1px solid #ddd;cursor:pointer}
  </style>
</head>
```

```
<body>
  <header>
    <h1>Tarea Switch — Viajes con JavaScript</h1>
    <p>El contenido se decide con <strong>switch</strong> según tus elecciones.</p>
    <button id="btn">Empezar</button>
  </header>

  <main class="panel" aria-live="polite" aria-atomic="true">
    <h2 id="titulo">Listo cuando tú quieras</h2>
    <p id="texto">Pulsa “Empezar”.</p>
    <div id="galeria" class="imgs" aria-label="Galería de imágenes"></div>
  </main>

  <script src="app.js"></script>
</body>
</html>
```

🧾 3) `app.js` (código tal cual)

```
const btn = document.getElementById("btn");
const titulo = document.getElementById("titulo");
const texto = document.getElementById("texto");
const galeria = document.getElementById("galeria");

btn.addEventListener("click", iniciar);

function iniciar() {
  limpiar();

  const quiere = confirm("¿Deseas jugar con JavaScript?");
  if (!quiere) {
    pintar(
      "Tú te lo pierdes!!!",
      "Has dicho que no. El JavaScript se queda triste (pero profesional).",
      [{ src: "img/nope.jpg", alt: "Imagen de despedida: no jugar con JavaScript" }]
    );
    return;
  }

  const destino = normalizar(prompt("Elige un destino: Francia, España, Portugal"));
  switch (destino) {
    case "francia":
      flujoFrancia();
      break;
    case "españa":
    case "espana":
      flujoEspana();
      break;
    case "portugal":
      flujoPortugal();
      break;
    default:
      pintar(
```

```
        "Destino no válido",
        "Debes escribir: Francia, España o Portugal.",
        []
      );
  }
}

function flujoFrancia() {
  const ciudad = normalizar(prompt("Francia: ¿Paris, Marsella o Lyon?"));
  switch (ciudad) {
    case "paris":
      pintar(
        "París",
        lorem4(),
        [
          { src: "img/paris1.jpg", alt: "Foto 1 de París" },
          { src: "img/paris2.jpeg", alt: "Foto 2 de París" },
          { src: "img/paris3.jpg", alt: "Foto 3 de París" }
        ]
      );
      break;

    case "marsella":
      pintar(
        "Marsella",
        lorem4(),
        [
          { src: "img/marsella1.jpeg", alt: "Foto 1 de Marsella" },
          { src: "img/marsella2.jpeg", alt: "Foto 2 de Marsella" },
          { src: "img/marsella3.jpeg", alt: "Foto 3 de Marsella" }
        ]
      );
      break;

    case "lyon":
      pintar(
        "Lyon",
        lorem4(),
        [
          { src: "img/lyon1.jpeg", alt: "Foto 1 de Lyon" },
          { src: "img/lyon2.jpeg", alt: "Foto 2 de Lyon" },
          { src: "img/lyon3.jpeg", alt: "Foto 3 de Lyon" }
        ]
      );
      break;

    default:
      pintar(
        "Ciudad no válida",
        "Debes escribir: Paris, Marsella o Lyon.",
        []
      );
  }
}
```

```
function flujoEspana() {
  const opcion = normalizar(prompt("España: ¿Playa o montaña?"));
  switch (opcion) {
    case "playa":
      pintar(
        "España — Playa",
        lorem4(),
        [
          { src: "img/playa1.jpeg", alt: "Foto 1 de playa" },
          { src: "img/playa2.jpeg", alt: "Foto 2 de playa" },
          { src: "img/playa3.jpg", alt: "Foto 3 de playa" }
        ]
      );
      break;

    case "montaña":
    case "montana":
      pintar(
        "España — Montaña",
        lorem4(),
        [
          { src: "img/montana1.jpeg", alt: "Foto 1 de montaña" },
          { src: "img/montana2.jpg", alt: "Foto 2 de montaña" }
        ]
      );
      break;

    default:
      pintar(
        "Opción no válida",
        "Debes escribir: Playa o Montaña.",
        []
      );
  }
}

function flujoPortugal() {
  const opcion = normalizar(prompt("Portugal: ¿Turismo, playa o montaña?"));
  switch (opcion) {
    case "turismo":
      pintar(
        "Portugal — Turismo",
        "Lisboa te espera. Callejeo, miradores y comida que no cabe en un commit.",
        [{ src: "img/lisboa.jpeg", alt: "Foto de Lisboa" }]
      );
      break;

    case "playa":
      pintar(
        "Portugal — Playa (Algarve)",
        "Algarve: texto + dos fotos, tal como pide el enunciado.",
        [
          { src: "img/algarve1.jpg", alt: "Foto 1 del Algarve" },
```

```
          { src: "img/algarve2.jpg", alt: "Foto 2 del Algarve" }
        ]
      );
      break;

    case "montaña":
    case "montana":
      pintar(
        "Portugal – Montaña",
        "Nazaret (Nazaré) y el santuario de Fátima.",
        [
          { src: "img/nazare.jpg", alt: "Foto de Nazaré" },
          { src: "img/fatima.jpeg", alt: "Foto del santuario de Fátima" }
        ]
      );
      break;

    default:
      pintar(
        "Opción no válida",
        "Debes escribir: Turismo, Playa o Montaña.",
        []
      );
  }
}

/* ---------- Helpers ---------- */

function pintar(t, p, imagenes) {
  titulo.textContent = t;
  texto.textContent = p;

  for (const im of imagenes) {
    const img = document.createElement("img");
    img.src = im.src;
    img.alt = im.alt;
    galeria.appendChild(img);
  }
}

function limpiar() {
  titulo.textContent = "";
  texto.textContent = "";
  galeria.innerHTML = "";
}

function normalizar(valor) {
  return (valor ?? "")
    .trim()
    .toLowerCase()
    .normalize("NFD")
    .replace(/[\u0300-\u036f]/g, "");
}
```

```
function lorem4() {
  return (
    "Lorem ipsum dolor sit amet, consectetur adipiscing elit. " +
    "Sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. " +
    "Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris. " +
    "Duis aute irure dolor in reprehenderit in voluptate velit esse."
  );
}
```

🧠 4) Comentario detallado del `app.js`

🔹 4.1. Captura de elementos del DOM

```
const btn = document.getElementById("btn");
const titulo = document.getElementById("titulo");
const texto = document.getElementById("texto");
const galeria = document.getElementById("galeria");
```

- `document.getElementById(...)` busca un elemento por su atributo `id` en el HTML.
- Si el elemento no existe o el script se ejecuta antes de cargar el HTML, devolvería `null`.
- En este ejercicio funciona porque el `<script src="app.js"></script>` está **al final del `<body>`**.

Uso de cada variable:

- `btn`: botón que inicia el juego.
- `titulo`: `<h2>` donde se mostrará el título de la opción elegida.
- `texto`: `<p>` donde se mostrará la descripción.
- `galeria`: `<div>` donde se insertarán las imágenes.

🔹 4.2. Evento de clic: punto de entrada

```
btn.addEventListener("click", iniciar);
```

- `addEventListener("click", ...)` registra una función para que se ejecute cuando el usuario haga clic.
- `iniciar` es el *handler* principal: desde ahí se lanza todo el flujo.

Ventaja:

- No se ejecuta nada hasta que el usuario interactúa (mejor control y menos errores).

🔹 4.3. Función `iniciar()`: arranque y primera decisión

```
function iniciar() {
  limpiar();
  const quiere = confirm("¿Deseas jugar con JavaScript?");
  ...
}
```

✅ **`limpiar()`**

- Se llama siempre al inicio para dejar la interfaz en “estado limpio”.
- Evita que se acumulen imágenes o textos de una ejecución anterior.

✅ `confirm(...)`

- Muestra una ventana con **Aceptar / Cancelar**
- Devuelve un boolean:
  - `true` si pulsa Aceptar
  - `false` si pulsa Cancelar

🔹 4.4. Caso NO: feedback + imagen

```
if (!quiere) {
  pintar(...);
  return;
}
```

- `!quiere` significa “si quiere es falso”.
- Se llama a `pintar(...)` para mostrar contenido.
- `return;` corta la ejecución: no se sigue preguntando destino.

Esto evita:

- Preguntas innecesarias
- Flujo incoherente (seguir el juego cuando el usuario dijo NO)

🔹 4.5. Elección de destino (switch principal)

```
const destino = normalizar(prompt("Elige un destino: Francia, España, Portugal"));
switch (destino) { ... }
```

✅ `prompt(...)`

- Abre una ventana para escribir texto.
- Devuelve:
  - una cadena (string)
  - o `null` si el usuario cancela

✅ `normalizar(...)`

Se aplica para permitir:

- “España”, “espana”, “ESPAÑA”, etc.
- Evita fallos por tildes o mayúsculas.

✅ `switch(destino)`

Controla las rutas:

- `"francia"` → `flujoFrancia()`
- `"españa"` o `"espana"` → `flujoEspana()`
- `"portugal"` → `flujoPortugal()`

- `default` → error y mensaje

🔹 4.6. Flujos por país (switch anidados)

🇫🇷 `flujoFrancia()`

- Pide ciudad
- `switch(ciudad)` con 3 casos
- Cada caso llama a `pintar()` con:
  - título
  - lorem
  - 3 imágenes

Punto clave:

- Las rutas `img/...` deben coincidir con el archivo real (`.jpg` / `.jpeg`).

🇪🇸 `flujoEspana()`

- Pide “playa” o “montaña”
- En “playa” se cargan 3 imágenes
- En “montaña” se cargan 2 imágenes

Se contemplan:

- `"montaña"` y `"montana"` por accesibilidad de teclado y compatibilidad (sin tildes).

🇵🇹 `flujoPortugal()`

- Pide “turismo”, “playa” o “montaña”
- “turismo” → 1 imagen (Lisboa)
- “playa” → 2 imágenes (Algarve)
- “montaña” → 2 imágenes (Nazaré + Fátima)

🔹 4.7. Función `pintar(...)`: renderizado dinámico

```
function pintar(t, p, imagenes) {
  titulo.textContent = t;
  texto.textContent = p;

  for (const im of imagenes) {
    const img = document.createElement("img");
    img.src = im.src;
    img.alt = im.alt;
    galeria.appendChild(img);
  }
}
```

✅ `textContent`

- Inserta texto seguro (sin interpretar HTML).

- Buena práctica frente a `innerHTML` cuando solo necesitas texto.

✅ Bucle `for...of`

- Recorre el array de objetos `imagenes`.
- Cada objeto tiene:
  - `src`: ruta de la imagen
  - `alt`: texto alternativo (accesibilidad)

✅ `document.createElement("img")`

- Crea un `<img>` real desde JavaScript.

✅ `appendChild`

- Inserta ese `<img>` dentro del contenedor `galeria`.

Punto de accesibilidad:

- `alt` es obligatorio para cumplir requisitos básicos.

🔹 4.8. Función `limpiar()`: reinicio de la UI

```
function limpiar() {
  titulo.textContent = "";
  texto.textContent = "";
  galeria.innerHTML = "";
}
```

- Limpia título y texto
- `galeria.innerHTML = ""` borra todas las imágenes insertadas antes

Si no existiera, al repetir el ejercicio:

- Las imágenes se irían sumando (error visual y de usabilidad)

🔹 4.9. Función `normalizar(...)`: robustez en inputs

```
function normalizar(valor) {
  return (valor ?? "")
    .trim()
    .toLowerCase()
    .normalize("NFD")
    .replace(/[\u0300-\u036f]/g, "");
}
```

Qué hace cada paso:

- `(valor ?? "")`:
  - si el prompt devuelve `null`, se convierte en `""` (evita error)
- `.trim()`:
  - elimina espacios al principio y final
- `.toLowerCase()`:

    - pasa a minúsculas para comparar en `switch`
- `.normalize("NFD")`:
    - separa caracteres con tildes en “letra + marca”
- `.replace(/[\u0300-\u036f]/g, "")`:
    - elimina esas marcas de tilde → “españa” pasa a “espana”

🔹 4.10. Función `lorem4()`: texto de relleno

```
function lorem4() {
  return ("Lorem ...");
}
```

- Devuelve un texto de ejemplo para simular un párrafo de unas 4 líneas.
- Se reutiliza en varias opciones para no repetir el mismo literal en cada caso.

✅ 5) Puntos críticos para que funcione a la primera

- El `<script src="app.js"></script>` debe ir **al final del** `<body>` o usar `defer`.
- Los nombres de archivo deben coincidir:
    - `paris2.jpeg` ≠ `paris2.jpg`
- Todas las imágenes deben tener `alt` (accesibilidad).
- `normalizar()` evita errores de usuario al escribir con tildes o mayúsculas.

```
Dataview (inline field '=='): Error:
-- PARSING FAILED -----------------------------------------------------

> 1 | ==
    | ^

Expected one of the following:

'(', 'null', boolean, date, duration, file link, list ('[1, 2, 3]'), negated
field, number, object ('{ a: 1, b: 2 }'), string, variable
```